Replicated Ambient Petri Nets

David de Frutos Escrig and Olga Marroquín Alonso

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid, Spain
{defrutos,alonso}@sip.ucm.es

Abstract Recently we have introduced Ambient Petri nets, as a multilevel extension of the Elementary Object Systems, that can be used to model the concept of nested ambients from the Ambient Calculus. Both mobile computing and mobile computation are supported by that calculus, and then by means of our Ambient Petri nets we get a way to introduce in the world of Petri nets these important features of nowadays computing. Nevertheless, our basic proposal does not yet provide the suitable background for the modeling of replication, one of the basic operators from the original calculus, by means of which infinite processes are introduced and treated in a very simple way. In this paper we enrich our framework by introducing that operator. We obtain a simple and nice model in which the basic nets are still static and finite, since the dynamics of the systems can be covered by the adequate notion of marking, where all the copies generated by the application of the replication operator will live together, without interfering in an inadequate way.

1 Introduction

Internet provides a computational infrastructure that spans the planet. Using it we get a nice support for both *mobile computing* and *mobile computation*. Mobile computing refers to virtual mobility (mobile software), while mobile computation refers to physical mobility (mobile hardware). Both kinds of mobility are elegantly modelled by the *Ambient Calculus* [3, 4].

The Ambient Calculus allows the movement of self-contained nested environments that include data and live computation. Those computational ambients, which are defined as bounded places where computation proceeds, have a *name*, a collection of *local processes*, and a collection of *subambients*. Ambients can move in and out of other ambients by means of *capabilities*, that are associated with ambient names. There exist three kinds of capabilities: *in* for entering an ambient, *out* for exiting an ambient, and *open* for opening up an ambient.

We are interested in translating this framework into the world of Petri nets. R.Valk has already studied his *Elementary Object Systems* (EOS) [9–11], that are composed of a system net where several object nets move. These object nets behave like ordinary tokens of the system net, that is, they lie in places and are moved by transitions, but also they may change their internal state (their marking), either when executing their own internal transitions or as a result of an interaction with a system transaction.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2658, pp. 774–783, 2003.

Elementary Object Systems provide two-level systems, but to cope with the features of the Ambient Calculus, we need arbitrary nesting. Then we had to unify both frameworks by defining *ambient Petri nets* [5], which allow the arbitrary nesting of object nets in order to model the arbitrary nesting of ambients.

Although the Ambient Calculus is a relatively simple model, we have found several technical (but interesting) difficulties related with the covering of different features of the calculus. Therefore, in order to give a clear and well motivated presentation, we have decided to introduce the model in an incremental way, by means of a series of papers, such that each one of them will focus on some of those features. So, in our first paper we have just considered the mobility primitives of the calculus and the parallel operator, which are enough to get a first notion of ambient net.

In this paper we enrich its definition in order to introduce the replication operator from the Ambient Calculus, $!P$, which generates an unbounded number of parallel copies of P. Besides, we will shortly explain how the restriction operation, $(\nu n)P$, which is used to introduce new names and limit their scope, interacts with that new operator.

Our final goal is to use ambient Petri nets to provide a denotational semantics for the Ambient Calculus. The way we follow, in order to encompass an algebraic formalism together with another Petri net based one, is that of the *Petri Box Calculus* (PBC) [1, 2, 6], which has been proved to be a suitable framework for these purposes. Therefore, our new model can be also interpreted as an extension of PBC that includes ambients [5].

In fact, we have a large experience in the development of PBC extensions. So, in [8] we have defined an elaborate timed extension, while in [7] we have presented a stochastic extension. By means of this new mobile version of PBC we try to introduce the ideas from the Ambient Calculus, which allows to model mobile systems in a simple but formally supported way. We hope that the developers of mobile systems who are familiar with Petri nets, will find in our formal model a tool to define those rather complicated systems, providing a formal basis for proving that the modelled systems fulfill their specifications.

1.1 Background

In this section we give a short overview of our basic model of ambient Petri nets.

A simple *ambient Petri net* is a finite collection of named Petri nets $A = \{n_1 : A_1, n_2 : A_2, \ldots, n_k : A_k\}$ for which we introduce a *location pair*, $\langle loc, open \rangle$, that defines both the current location of each component net, $A_i = (P_i, T_i, W_i)$ with $i \geq 2$, and its (open or closed) state. Intuitively, nets $\{A_2, \ldots, A_k\}$ can be seen as net tokens that move along the full set of places of A, thus representing the movement of ambients. As a consequence, it is possible to find in the places of an ambient Petri net both ordinary and high-level tokens. By unfolding the latter we obtain the collection of nested marked ordinary nets that constitute the ambient net, in which we say that n_1 is the *root net*.

In order to adequately support the mobility of ambients due to the execution of capabilities, each ambient Petri net has two kinds of transitions. Besides

the ordinary ones there is a set of high-level transitions, that we call *ambient transitions*, $\mathcal{A}mb(A) = \{In\ n_i, Out\ n_i, Open\ n_i \mid i \in \{2, \ldots, k\}\}$. Those ambient transitions are used for controlling the movement of the object tokens in A. Their firing is synchronized with the firing of the transitions in the component nets labelled by elements in $\mathcal{C} = \{in\ n_i, out\ n_i, open\ n_i \mid i \in \{2, \ldots, k\}\}$, thus modifying both the location of the modified component and the internal state of the triggering low-level token transition which represents the capability to move that component.

In the Ambient Calculus, the entry capability can be used by a process $in\ m.P$, to instruct the surrounding ambient to enter into a sibling ambient named m, as stated by the reduction rule $n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]$. In the case of ambient Petri nets we say that two net tokens are siblings if they are located in places of the same component. The firing of $(In\ n_i, in\ n_i)$ pairs will move a component net into another, but since we need the particular place where the token will be allocated, we will provide together with the capability the name of that place, thus having some $(in\ n_i, p_i)$ as the label of the corresponding transition.

Concerning the exit transitions, modelled in the Ambient Calculus by the capabilities $out\ m$, they produce the exit of the surrounding ambient of $out\ m.P$ from its containing ambient m, such as the corresponding reduction rule shows: $m[n[out\ m.P|Q]|R] \rightarrow m[R]|n[P|Q]$. Then, in the ambient Petri net model whenever a net token n_j located at n_i may fire an $out\ n_i$ transition, we can fire the ambient transition $Out(n_i, n_j)$ by moving n_j into the location of n_i.

Finally, ambients can be open (and destroyed) by using an opening capability, $open\ m$. Thus, $open\ m.P$ provides a way of dissolving the boundary of an ambient named m located at the same level that this process, according to the rule $open\ m.P|m[Q] \rightarrow P|Q$. The firing of pairs of the form $(Open\ n_i, open\ n_i)$ has the same effect in an ambient Petri net that the one described. Nevertheless, since we are interested in an static description of ambient nets, we do not remove the open net, but just attach to it a label that describes its (open) state. In this way, its contents will be treated in the future as parts of the containing component.

The execution of ordinary transitions, which are not labelled by capabilities, follows the firing rule for the ordinary Petri nets: The tokens in the preset places of the involved transition are consumed, and instead new tokens are added into the postset places.

As a consequence of mixing both ordinary and ambient transitions, we obtained a new class of Petri nets suitable for modeling mobile agents based systems. Nevertheless, this new framework does not support any mechanism of replication, which is widely used in order to represent replication of services. With this purpose, the Ambient Calculus provides expressions of the form $!P$ that represent an unbounded number of parallel replicas of P, and whose behaviour is reflected by the structural congruence relation between $!P$ and $P|!P$: $!P \equiv P|!P$.

The extension of the ambient nets with such a mechanism results in the definition of *Replicated Ambient nets*, which are described in the following section.

2 Formal Definitions

In order to define replicated ambient Petri nets we extend the ambient Petri nets in [5] to include the elements that will make possible the translation of the replication operator from the Ambient Calculus. We will motivate each necessary extension by means of a simple example. In each case we will study a term from our calculus, which combines operators from both the Ambient Calculus and PBC.

This calculus mixes together capabilities and ordinary actions that belong to a countable alphabet of labels $\mathcal{A}$, and provides some operators to combine processes in a compositional way. Amongst those operators, the new language includes the sequential composition ($\_;\_$), the parallel composition ($\_|\_$) and the synchronization ($\_\,\mathtt{sy}\,a$), all inherited from PBC, together with the replication operator ($!\_$).

The sequential composition is a generalized form of the prefix operator. In the parallel composition $P|Q$ the involved actions are independently executed by P and Q, either sequentially or concurrently, but without any communication between them. Therefore, if $P \xrightarrow{\Gamma} P'$ and $Q \xrightarrow{\Delta} Q'$ then $P|Q \xrightarrow{\Gamma+\Delta} P'|Q'$, where $\Gamma, \Delta \in \mathcal{M}(\mathcal{A})$. Finally, in order to support synchronization, we assume the existence of a bijection $\widehat{\ }\colon \mathcal{A} \longrightarrow \mathcal{A}$, called *conjugation*, by means of which we associate to each label $a \in \mathcal{A}$ the corresponding $\hat{a} \in \mathcal{A}$. This function must satisfy that $\forall a \in \mathcal{A} \quad \hat{a} \neq a \wedge \hat{\hat{a}} = a$. Then synchronization, which is never forced, is modeled by means of pairs of conjugated actions in such a way that if $P\,\mathtt{sy}\,a \xrightarrow{\{\alpha+a\}+\{\beta+\hat{a}\}+\Gamma} P'\,\mathtt{sy}\,a$ then $P\,\mathtt{sy}\,a \xrightarrow{\{\alpha+\beta\}+\Gamma} P'\,\mathtt{sy}\,a$. By applying this operator over terms of the form $P = P_1|P_2$ we obtain the usual communication mechanism between processes, although, as stated above, it is not mandatory, since $P\,\mathtt{sy}\,a$ can still mimic the behaviour of P: Whenever $P \xrightarrow{\Gamma} P'$ we have also $P\,\mathtt{sy}\,a \xrightarrow{\Gamma} P'\,\mathtt{sy}\,a$.

Example 1. Let us consider the term $!a$. In order to represent the creation of a new clone of the body of the replication, as stated by the expansion $!a \equiv a|!a$, we introduce a τ-transition connected to the entry place of the process (Figure 1(a)). In this way, whenever a new copy of a is needed, the τ-transition is fired generating two new tokens: One of them will occupy the precondition of a, which allows to initiate a new execution of that body expression, while the other one stays in the precondition of the τ-transition awaiting for a new replication of the process. In our graphical representations we will omit τ-transitions by replacing them with arcs that directly connect the involved places. As a consequence, for the net in Figure 1(a) we would get that in Figure 1(b).

By applying this definition we would lose safeness of the markings, since by firing the initial τ-transition one can create as many tokens as desired in the entry places of the net corresponding to the body of the operator. Nevertheless, this is necessary in order to represent in the same net the parallel execution of an unbounded number of replicas of the net. However, in some cases it is important to separate the execution of those copies, to adequately translate the semantics of the replication operator.

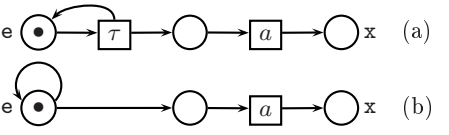

Fig.1. (a) Representing replication (b) Compact representation (omiting τ's)

Example 2. Let us consider the term $!((a|\hat{a})\mathtt{sy}\ a)$. If we apply the simple translation from the previous example we would get the net in Figure 2(a), once we just remove all the tokens in its places. Then, if we expand twice the replication operator we would get the expression $((a|\hat{a})\mathtt{sy}\ a)|((a|\hat{a})\mathtt{sy}\ a)|!((a|\hat{a})\mathtt{sy}\ a)$, which in the net will produce a couple of tokens in each of the entry places of the encoding of the term $(a|\hat{a})\mathtt{sy}\ a$. How do we represent the joint firing of a from the first copy and $\hat{a}$ from the second?. If the textual expression performs both actions we obtain a term in which we cannot fire the synchronizing transition without a new expansion of the replication operator:

$$\begin{aligned}((a|\hat{a})\mathtt{sy}\ a)|((a|\hat{a})\mathtt{sy}\ a)|!((a|\hat{a})\mathtt{sy}\ a) &\xrightarrow{a} (\hat{a}\mathtt{sy}\ a)|((a|\hat{a})\mathtt{sy}\ a)|!((a|\hat{a})\mathtt{sy}\ a)\\ &\xrightarrow{\hat{a}} (\hat{a}\mathtt{sy}\ a)|(a\mathtt{sy}\ a)|!((a|\hat{a})\mathtt{sy}\ a)\end{aligned}$$

Instead, in the net representation we can use the available tokens from the two replicas to fire the synchronizing transition. We conclude that it is necessary to personalize the tokens from each activation of a replication, to prevent from consuming tokens of several copies in the firing of a transition, that should be restricted to single copies of the body. This is done by labelling tokens in the scope of a replication operator with a natural number that identifies the serial number to which it corresponds. So we get the representation in Figure 2(b).

But in order to cover the case of nested replicated operators we still need another generalization.

Example 3. Let us consider the term $!(a;!b)$. Each activation of the main replication operator generates a term $a;!b$, where still a collection of different copies of b would be generated. This behaviour is reflected by the following computation, in which we can execute b actions belonging to different replicas of the term $a;!b$:

$$\begin{aligned}!(a;!b) &\equiv (a;!b)|(a;!b)|!(a;!b) \xrightarrow{a} !b|(a;!b)|!(a;!b) \xrightarrow{a} !b|!b|!(a;|b)\\ &\equiv (b|b|!b)|(b|!b)|!(a;!b) \xrightarrow{b} (b|b|!b)|!b|!(a;!b)\end{aligned}$$

The solution to link each replica of b with the corresponding clone of $a;!b$, that is, to individualize the involved tokens in a simple and satisfactory way, is to

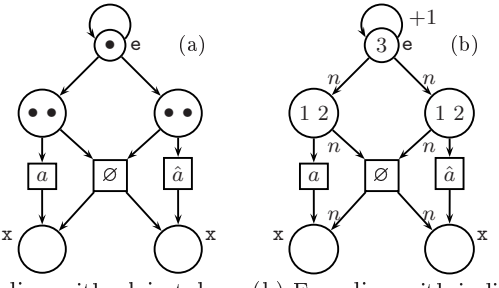

Fig.2. (a) Encoding with plain tokens (b) Encoding with individualized tokens

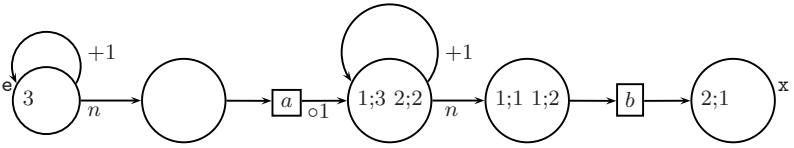

Fig.3. Encoding with sequences of natural numbers

label the tokens with a sequence of natural numbers, with an element for each nested replicated operator. Thus, for the given computation we would get the marked net in Figure 3.

For any place in a subnet which is not in the scope of any replication operator, the tokens in it would be labelled with the empty sequence ϵ.

With all this in mind we can give the definitions for replicated ambient Petri nets. Following the ideas from the Petri Box Calculus we distinguish static (unmarked) nets and dynamic (marked) ones. The first constitute the fixed architecture of the corresponding system, which remains the same along its execution. Therefore, static replicated ambient Petri nets are defined as in the simple non-replicated case (see [5]), although now component nets can contain τ-transitions which will be used to generate the copies of the replicated subnets of the system. Besides, we need some annotations in the arcs to cope with the adequate labelling of the tokens involved in the firing of some transitions. Therefore, we start with a given set of ambient names $\mathcal{N}$, and then we have:

Definition 4 (Component net). *A* ***component net*** *is an ordinary Petri net* $N = (P, T, W)$*, where:*

- P *is the set of places, which is partitioned into three subsets:* E *of* ***entry places****,* X *of* ***exit places*** *and* I *of* ***internal places****,*
- T *is the set of transitions, disjoint from* P*, that contains two subsets:* $\mathcal{I}nt(T)$*, composed of* ***internal transitions****, and* $\mathcal{A}mb(T)$*, constituted by* ***ambient transitions*** *which are labelled by pairs of the form* (cap, n) *with* $cap \in \{in, out, open\}$ *and* $n \in \mathcal{N}$*,*
- $W \subseteq (P \times T) \cup (T \times P)$ *is the set of connection arcs. Some of its elements could have an annotation in the set* $L \cup \{+1, \circ 1\}$*, where* L *is a set of variables to denote labels of tokens,* $+1$ *represents the addition of one unit to the last term of the label of the involved token, and* $\circ 1$ *represents the concatenation with a new* 1*.*

Example 5. The (single) component net for the term $!(a; !b)$ would be the net in Figure 3, if we just remove all the tokens from its places.

Definition 6 (Static replicated ambient Petri net). *A* ***static replicated ambient Petri net*** *is a finite collection of named component nets* $A = \{n_1 : A_1, n_2 : A_2, \ldots, n_k : A_k\}$ *where* $k \geq 1$*,* $n_1 = root$ *and* $\forall i, j \in \{1, \ldots, k\}\ n_i \in \mathcal{N} \wedge (i \neq j \Rightarrow n_i \neq n_j)$*.*

In the following, we will call to A_1 the root net of A and $\mathcal{N}(A) = \{n_2, \ldots, n_k\}$ will denote the ambient nets that can be used in the system. For each one of them we have the corresponding component which defines the behaviour of each

copy of this ambient. In ordinary ambient nets we cannot replicate any term, which is captured by the fact that each component can be only performed once, but now we can begin the execution of a new copy of any ambient whenever a τ-transition is fired, reflecting the replication of the corresponding subnet. As a consequence, in a dynamic replicated ambient Petri net we may have several copies of some ambients. Each one of those replicas can be identified by the set of tokens that are involved in its execution.

Definition 7 (Ambient copies). *Given a replicated ambient net A, a set of* ***ambient copies*** *for it is defined by means of a function $Copies : \mathcal{N}(A) \to \mathcal{P}_f(\mathbb{N}^*)$ which verifies that $\forall c, c' \in Copies(n)\ |c| = |c'|$.*

As stated above, the values in $Copies(n)$ denote the set of labelled tokens that have been used to fire a new copy of the ambient. Since the replicated ambient will be situated in a fixed place of the system, then all the tokens that can visit that place will have the same length, this is why we impose that all the copies of each ambient will be labelled by strings of the same length. More exactly, any replica of an ambient n will be denoted by a pair composed of the name of this ambient and the label c of the token that is involved in its activation.

The behaviour of the copies of an ambient is not interconnected, so the location of each ambient copy of $n \in \mathcal{N}(A)$ must be defined independently by means of the location pair corresponding to A. Due to the arbitrary nesting of component nets, an ambient copy of $n \in \mathcal{N}(A)$, $\langle n, c\rangle$, could be located in a place p' of an ambient copy of $n' \in \mathcal{N}(A)$, $\langle n', c'\rangle$, which is denoted by $loc(\langle n, c\rangle) = (\langle n', c'\rangle, p')$, and because of the performance of the *open* capability, the boundary of any ambient replica could be broken, which is denoted by $open(\langle n, c\rangle) = true$.

Definition 8 (Location pair for ambient copies). *Given a static replicated ambient Petri net A and a set of ambient copies for it defined by the function Copies, we define a* ***location pair*** *for them as a pair of partial functions $\langle loc, open\rangle$ with $loc : \mathcal{N}(A) \times (\mathbb{N}^* \cup \{0\}) \to ((\mathcal{N}(A) \cup root) \times (\mathbb{N}^* \cup \{0\})) \times P$ verifying*

- $loc(\langle n, c\rangle) \downarrow \iff (c \in Copies(n) \vee c = 0)$,
- $(c \in \mathbb{N}^* \wedge loc(\langle n, c\rangle) = (\langle n', c'\rangle, p')) \Longrightarrow (n' = root \wedge c' = \epsilon) \vee c' \in Copies(n')$,
- $loc(\langle n, 0\rangle) = (\langle n', c'\rangle, p') \Longrightarrow (n' = root \wedge c' = \epsilon) \vee (n' \neq root \wedge c' = 0)$,

and $open : \mathcal{N}(A) \times \mathbb{N}^* \to Bool$, with $open(\langle n, c\rangle) \downarrow \iff c \in Copies(n)$.

In this way, if n is the name of a component net, then we have two kinds of associated ambients: $\langle n, 0\rangle$ represent the original ambient from which we can generate copies and it will be located in some place of the system. Whenever a token labelled by c arrives to that place, we are able to generate the corresponding copy $\langle n, c\rangle$, whose execution starts. The original ambient token will remain at the same place, ready to generate new copies when new generating tokens will arrive to the involved place. These generated replicas will be the active net tokens of the system. In particular, they can be moved by means of the corresponding ambient transition. Instead, the original ambients are static: Neither they are executed nor moved along the system.

Definition 9 (Located replicated ambient Petri net). *A **located replicated ambient Petri net** is a static replicated ambient net for which we have defined the location pair corresponding to its full set of ambient copies.*

Definition 10 (Dynamic replicated ambient Petri net). *A **dynamic replicated ambient Petri net** is a located replicated ambient net for which we have defined an ordinary marking $M : P \longrightarrow \mathcal{M}_f(\mathbb{N}^*)$, where P is the full set of places of the ambient net, that is, $P = \bigcup_{i=1}^{k} P_i$ with $A_i = (P_i, T_i, W_i)$.*

All the markings of the different copies of each replicated ambient are put together. This is not a problem since the ordinary tokens for the execution of a copy $\langle n_i, c_i \rangle$ will be those tokens in the places of P_i labelled by sequences extending the sequence c_i.

Markings of replicated ambient Petri nets consist of two components: the function *Copies*, which defines the set of ambient tokens in the net, and M, which defines the ordinary marking for each ambient token.

Definition 11 (Initial marking). *The **initial marking** of a located replicated ambient Petri net A, $\langle Copies_{init}, M_{init} \rangle(A)$, is that with $Copies_{init}(n_i) = \varnothing$ $\forall n_i \in \mathcal{N}(A)$ and where only the entry places of A_1 are marked, that is, $\forall p \in E(A_1)$ $M_{init}(p) = \{\epsilon\}$ and $\forall p \in P \backslash E(A_1)$ $M_{init}(p) = \varnothing$.*

Definition 12 (Activation rule). *Whenever we have a marking $\langle Copies, M \rangle$ of a dynamic replicated ambient net such as for some $p_j \in P$ and $c \in \mathbb{N}^*$ verifies that $M(p_j)(c) > 0$, and there exists some n_i such that $loc(\langle n_i, 0 \rangle) = (\langle n_j, 0 \rangle, p_j)$, we can fire an internal activation transition which consumes an ordinary token in p_j labelled by c, producing a new copy of n_i (then we have $Copies'(n_i) = Copies(n_i) \cup \{c\}$), whose entry places will be also marked by tokens labelled with the same sequence c.*

Dynamic ambient tokens will move along the system by means of the firing of those transitions labelled by ambient operations. As an example of the different firing rules for these high-level transitions associated to those transitions expressing capabilities, next we give the rule for the entry operation.

Definition 13 (Entry operation). *Let $c_j \in Copies(n_j)$ be such that under the ordinary marking for $\langle n_j, c_j \rangle$ we can fire a transition $t_j \in T_j$ labelled by $(in\ n_i, p_i)$. If $loc(\langle n_j, c_j \rangle) = (\langle n_k, c_k \rangle, p_k)$ and there exists some $c_i \in Copies(n_i)$ with $loc(\langle n_i, c_i \rangle) = (\langle n_k, c_k \rangle, p'_k)$ then we can fire the high-level transition associated to t_j, getting as reached state of A that defined by the following changes in M:*

- *The local marking of A_j (more exactly, that of its c_j-copy) changes as the ordinary firing rule for t_j says.*
- *The location of the replica $\langle n_j, c_j \rangle$ changes, getting $loc(\langle n_j, c_j \rangle) = (\langle n_i, c_i \rangle, p_i)$.*

Note that the different replicas of an ambient n_i could move in different directions, and therefore they could stay at different locations of the same system. Nevertheless, the original copies (denoted by $\langle n, 0 \rangle$) , which are static by definition, will remain at their original location without ever moving.

3 Replication and the Restriction Operation

The restriction operator νn is introduced in the Ambient Calculus as a mechanism for defining internal names which cannot be used in the outer environment, neither by chance nor if we have known those names in an inappropriate way. The only way to get internal names is by a willingly communication of the internal process, which would produce the extrusion of the restriction operator. In this way a simple but powerful security mechanism is formalized.

For instance, in the term $m[in\ n]|(\nu n)n[a]$ we use the same ambient name n inside m, but from there we cannot reach the so named ambient. In fact, restricted names can be renamed without any change in the semantics of the system, since the Ambient Calculus identifies processes up to renaming of bound names. Therefore, process $m[in\ n]|(\nu n)n[a]$ is identical to $m[in\ n]|(\nu p)p[a]$.

Restriction can be treated by means of simple renamings if we consider systems without recursion and replication. But things are rather more complicated if, as in this paper, the replication operator is allowed.

For instance, if we consider the process $!(\nu n)(n[a]|m[in\ n])$ we have different internal ambient names in each of the copies of the system, due to the fact that replication creates new names ($!(\nu n)P \not\equiv (\nu n)!P$). As a consequence, each single copy should use each concrete name, although the behaviour of all the replicas is the same, once we take into account the corresponding renaming. Then it is possible to preserve the static definition of systems, including a fixed collection of ambient names and net components. In order to support the dynamic creation of new names, we use the fact that they can be seen as (different) copies of the original ones.

We have found that the structured labels of the tokens provide a nice way to support the sets of restricted names. So we only have to change the set $\mathbb{N}^*$ into $(\mathcal{P}(\mathcal{N}) \times \mathbb{N})^* \times \mathcal{P}(\mathcal{N})$, where the indexes associated to applications of the replication operator are intercalated with the sets of ambient names which are composed of the names that are restricted before and after the application of each replication operator.

For instance, for the process $(\nu n)!P$ we would get labels such as $\{n\}1\varnothing$, while $!(\nu n)P$ would produce instead $\varnothing 1\{n\}$. Then, if a copy of the ambient n is activated with these tokens, we would obtain a renamed copy whose name is n_c where c is the prefix of the place of the sequence where n is restricted. So in the examples above we would get n_ϵ and n_1. The new name would be taken into account to avoid unsuitable accesses to the restricted name. More on the subject in the next forthcoming paper, completely devoted to the study of the restriction operator.

4 Conclusions and Future Work

In this paper we have extended the basic model of ambient Petri nets in order to support the replication operator from the Ambient Calculus. We have seen that even if replication produced the dynamic creation of nets, which would represent

the copies of the replicated ambients, we can still base our definitions on a static system describing the set of basic components of its architecture. Net tokens in the markings of these systems represent the dynamic activation of ambients.

We have seen that although we had to afford some technical difficulties a nice solution was still possible. Besides, this solution only has to be slightly modified to cope with the interrelation between the replication and the restriction operators.

In our opinion it is very important to give the designers who are familiar with Petri nets a (relatively) simple extension by means of which they will be able to develop mobile systems, whose correctness could be formally proved.

We are currently working on the remaining features of the Ambient Calculus and PBC in order to get the full framework in which we are interested.

Acknowledgements.
Research supported by CICYT project Desarrollo Formal de Sistemas Basados en Agentes Móviles (TIC 2000-0701-C02-01)

References

1. E. Best, R. Devillers and J. Hall. *The Petri Box Calculus: A New Causal Algebra with Multi-label Communication.* Advances in Petri Nets 1992, LNCS vol.609, pp.21-69. Springer-Verlag, 1992.
2. E. Best, R. Devillers and M. Koutny. *Petri Net Algebra.* EATCS Monographs on Theoretical Computer Science Series. Springer-Verlag, 2001.
3. L. Cardelli. *Abstractions for Mobile Computation.* Secure Internet Programming: Security Issues for Mobile and Distributed Objects, LNCS vol.1603, pp.51-94. Springer-Verlag, 1999.
4. L. Cardelli. *Mobility and Security.* Proceedings of the NATO Advanced Study Institute on Foundations of Secure Computation, pp.3-37. IOS Press, 2000.
5. D. Frutos Escrig and O. Marroquín Alonso. *Ambient Petri Nets.* Submitted for publication.
6. M. Koutny and E. Best. *Operational and Denotational Semantics for the Box Algebra.* Theoretical Computer Science 211, pp.1-83, 1999.
7. H. Maciá, V. Valero and D. Frutos Escrig. *sPBC: A Markovian Extension of Finite Petri Box Calculus.* Petri Nets and Performance Models PNPM'01, pp.207-216. IEEE Computer Society, 2001.
8. O. Marroquín Alonso and D. Frutos Escrig. *Extending the Petri Box Calculus with Time.* Applications and Theory of Petri Nets 2001, LNCS vol.2075, pp.303-322. Springer-Verlag, 2001.
9. R. Valk. *Petri Nets as Token Objects: An Introduction to Elementary Object Nets.* Applications and Theory of Petri Nets 1998, LNCS vol.1420, pp.1-25. Springer-Verlag, 1998.
10. R. Valk. *Relating Different Semantics for Objects Petri Nets, Formal Proofs and Examples.* Technical Report FBI-HH-B-226, pp.1-50. University of Hamburg, Department for Computer Science, 2000.
11. R. Valk. *Concurrency in Communicating Object Petri Nets.* Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets, LNCS vol.2001, pp.164-195. Springer-Verlag, 2001.